Practical – 5

Aim: Dynamic Programming Approach

5.1) Let S be a collection of objects with profit-weight values. Implement the 0/1 knapsack problem for S assuming we have a sack that can hold objects with total weight W.

Program Code:

```
import java.util.*;
class ProfitWeight {
    int profit;
    int weight;

    public ProfitWeight(int profit, int weight) {
        this.profit = profit;
        this.weight = weight;
    }
}
public class prac {
    public static int knapsack(ProfitWeight[] objects, int W) {
        int n = objects.length;
        int[][] dp = new int[n + 1][W + 1];
        for (int i = 1; i <= n; i++) {
            for (int j = 0; j <= W; j++) {
                if (objects[i - 1].weight <= j) {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i - 1][j - objects[i - 1].weight] + objects[i -
1].profit);
                } else {
```

```
                    dp[i][j] = dp[i - 1][j];
                }
            }
        }
        return dp[n][W];
    }
    public static void main(String[] args) {
        ProfitWeight[] objects = {
            new ProfitWeight(60, 10),
            new ProfitWeight(100, 20),
            new ProfitWeight(120, 30)
        };
        int W = 50;
        System.out.println("Maximum profit: " + knapsack(objects, W));
    }
}
```

Output:

```
PS D:\Probin's Work\Extra> javac prac.java
PS D:\Probin's Work\Extra> java prac
Maximum profit: 220
PS D:\Probin's Work\Extra>
```

5.2) Implement a program to print the longest common subsequence for the following strings.

Test Case	String1	String2
1	ABCDAB	BDCABA
2	EXPONENTIAL	POLYNOMIAL
3	LOGARITHM	ALGORITHM

Program Code:

```
import java.util.*;
public class prac {
    public static String lcs(String s1, String s2) {
        int m = s1.length();
        int n = s2.length();
        int[][] dp = new int[m + 1][n + 1];

        // Build the dp array
        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
                if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
                    dp[i][j] = dp[i - 1][j - 1] + 1;
                } else {
                    dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);
                }
            }
```

```
        }

        // Build the LCS string
        StringBuilder lcs = new StringBuilder();
        int i = m, j = n;
        while (i > 0 && j > 0) {
            if (s1.charAt(i - 1) == s2.charAt(j - 1)) {
                lcs.insert(0, s1.charAt(i - 1));
                i--;
                j--;
            } else if (dp[i - 1][j] > dp[i][j - 1]) {
                i--;
            } else {
                j--;
            }
        }
        return lcs.toString();
    }

    public static void main(String[] args) {
        String s1 = "ABCDAB";
        String s2 = "BDCABA";
        System.out.println("Longest Common Subsequence: " + lcs(s1, s2));
    }
}
```

Output:

```
PS D:\Probin's Work\Extra> javac prac.java
PS D:\Probin's Work\Extra> java prac
Longest Common Subsequence: BDAB
PS D:\Probin's Work\Extra>
```

5.3) Given a chain < A1, A2,...,An> of n matrices, where for i=1,2,...,n matrix Ai with dimensions. Implement the program to fully parenthesize the product A1,A2,...,An in a way that minimizes the number of scalar multiplications. Also calculate the number of scalar multiplications for all possible combinations of matrices.

Test Case	n	Matrices with dimensions
1	3	A1: 3*5, A2: 5*6, A3: 6*4
2	6	A1: 30*35, A2: 35*15, A3: 15*5, A4: 5*10, A5: 10*20, A6: 20*25

Program Code:

```
import java.util.*;
public class prac {

    public static int matrixChainOrder(int[] dimensions) {
        int n = dimensions.length - 1;
```

```
        int[][] dp = new int[n][n];

        for (int i = 0; i < n; i++)
            dp[i][i] = 0;

        for (int len = 2; len <= n; len++) {
            for (int i = 0; i < n - len + 1; i++) {
                int j = i + len - 1;
                dp[i][j] = Integer.MAX_VALUE;
                for (int k = i; k < j; k++) {
                    int cost = dp[i][k] + dp[k + 1][j] + dimensions[i] * dimensions[k + 1] *
dimensions[j + 1];
                    if (cost < dp[i][j])
                        dp[i][j] = cost;
                }
            }
        }
        return dp[0][n - 1];
    }

    public static void main(String[] args) {
        int[] dimensions1 = {3, 5, 6, 4};
        System.out.println("Minimum scalar multiplications: " +
matrixChainOrder(dimensions1)+ " for example1");

        int[] dimensions2 = {30, 35, 15, 5, 10, 20, 25};
        System.out.println("Minimum scalar multiplications: " +
matrixChainOrder(dimensions2)+" for example2");
```

```
    }
}
```

Output:

```
PS D:\Probin's Work\Extra> javac prac.java
PS D:\Probin's Work\Extra> java prac
Minimum scalar multiplications: 162 for example1
Minimum scalar multiplications: 15125 for example2
PS D:\Probin's Work\Extra>
```

Conclusion: From this practical I learned about the concept of Dynamic Programming.

Staff Signature:

Grade:

Remarks by the Staff: